M5Stack Core2 LCD Tutorial

by Bryan A. “CrazyUncleBurton” Thompson

Last Updated 2/21/2026

Sensor

M5Stack Unit ENV III Temp/Pressure/Humidity sensor array contains:

SHT3X for temp/humidity

QMP6988 for temp/pressure/altitude

Sensor Cable

4 wire Grove connector

Sensor Cable Connections

Connect cable to red port on sensor. Connect cable to red port on M5Stack Tab5.

USB C Cable – USB C on the Tab5 end. Whatever will plug into your computer on the other end.

Computer – VS Code and its extensions support Windows, Mac and Linux. This tutorial assumes a Windows 10/11 machine.

About the Project

This is a sample project from the manufacturer, M5Stack. I modified it to write to the LCD instead of the serial port. We also changed the metric to the Imperial Units of Freedom that allowed us to land on the moon.

Testing

1. Connect a USB cable from the computer that VS Code is running on to the M5Stack microcontroller.
2. Connect a cable from the red port on the microcontroller to the red port on the ENV III sensor.
3. If the microcontroller is not on, press the white button on the side once to turn it on.
4. In VS Code, click the PlatformIO Upload icon in the bottom toolbar. The hover text will be PlatformIO: Upload.
5. The project will compile. When it's ready to upload, we see lines like "Connecting…" and "Writing…". Wait for the line that says SUCCESS in green text before proceeding.
6. When finished, the microcontroller will reboot.
7. When the microcontroller reboots, watch the display. You should see a one-time splash screen that says M5Stack Core2 and then Starting…
8. After about one second (a delay inserted to let the user see the splash screen), we see the first update from the sensors.
9. One second later (again, to let people read the screen), it refreshes the data and updates the screen.

Program Explanation

We started with the program from the manufacturer as an example for using their sensor. It only output data to the LCD.

These lines are libraries provided by the microcontroller manufacturer and sensor manufacturer to support that hardware.

#include <M5Unified.h>

#include "M5UnitENV.h"

See the references below on how to use M5GFX. We instantiated that library with the name "display":

M5GFX& display = M5.Display;

This line enables canvas functionality, as our data will change every second or so.

M5Canvas canvas(&display);

These names instantiate the sensor libraries with the names SHT3X and QMP6988:

SHT3X sht3x;

QMP6988 qmp;

We added some variables to hold the current values of the sensors and the pins we're connecting the I2C sensor to, and some variables to configure the size of text on the screen and a really convulated way to represent the degree symbol we'll use for the temperature units:

float shtTempF = 0.0f;

float shtHumidity = 0.0f;

float qmpTempF = 0.0f;

float qmpPressureInHg = 0.0f;

float qmpAltitudeFt = 0.0f;

bool shtPresent = false;

bool qmpPresent = false;

const int i2cSdaPin = 32;

const int i2cSclPin = 33;

const int textSize = 2;

const int lineHeight = 40;

const char degreeSymbol[] = { static_cast<char>(0xC2), static_cast<char>(0xB0), '\0' };

Next we create the function drawDashboard(). drawDashboard() draws the next screen update to the canvas in the background. It isn't being called yet – it has to be defined in the program before we update it.

These lines clears the canvas and sets the size of the text we're writing to the canvas:

canvas.fillScreen(TFT_BLACK);

canvas.setTextSize(textSize);

The next line checks to see if the sensor is present. If so, we proceed. If not, we wait for the sensor to be connected:

if ((shtPresent)&&(qmpPresent)) {

This line sets the position on the canvas where the next text will be printed:

canvas.setCursor(x, y);

In this line we see a printf with variable substitution:

canvas.printf("Temp 1: %.2%sF", shtTempF, degreeSymbol);

This will print some formatted text, in this case “Temp 1: “. Then it prints the contents of the variable temp, which contains the value of the temperature measured from the SHT3X sensor. The “%.2f” part formats the floating point variable with two digits to the right of the decimal. Then it prints the units, “°F”.

This moves the cursor down to the next row.

y += lineHeight;

This pattern repeats for the five lines of data to be displayed. We added some more lines to print the other variables that the sensor library created and updated for us, but they’re not really different from what we did above, except for this line:

canvas.printf("Humidity: %.2f%% rH", shtHumidity);

This line prints “Humidity: “ and then the formatted contents of the sht3x.humidity variable, and then prints a “%” sign (the units of humidity), and then prints “ rH” (relative humidity). The “%.2f” part formats the floating point variable with two digits to the right of the decimal. Then it prints the units, “°F”. The “%%” part is doubled because printf treats a single % sign as formatting. The double %% tells printf we mean to print a literal % sign here.

Repeat for the other lines which print sensor data to the canvas.

This line moves the screen we created on the canvas above to the LCD all at once:

canvas.pushSprite(0, 0);

The setup function runs once at microcontroller power-up and restart:

These lines start the M5Unified library which supports the microcontroller:

auto cfg = M5.config();

M5.begin(cfg);

This line starts the Arduino I2C service for Wire1, which is the first alternate I2C bus:

Wire1.begin(i2cSdaPin, i2cSclPin);

These lines configure the display and canvas:

display.setRotation(1); - This sets which way is up on the LCD.

canvas.setColorDepth(16); - This sets the number of bits of color info on the LCD. This is RGB565 – 5 bits of Red, 6 bits of Green, and five bits of Blue.

canvas.createSprite(display.width(), display.height()); - These lines create the canvas, which is the background area where we build the screen. It won't be displayed until later.

canvas.setTextColor(TFT_WHITE, TFT_BLACK); - This sets text color to white, and black background color to white.

canvas.setFont(&fonts::efontJA_16_b); - This is a built-in font that supports Unicode symbols

canvas.setTextSize(textSize); - This sets the size of text we are printing to the screen.

canvas.setTextWrap(false, false); - This says not to wrap the text if it prints off the end of the line

These lines draw the initial splash screen:

canvas.fillScreen(TFT_BLACK); - Clears the screen

canvas.drawCenterString("M5Stack Core2", display.width() / 2, display.height() / 2 - 60); - This prints "M5Stack Core2", centered on the display minus a bit to offset it from the line below.

canvas.drawCenterString("Starting...", display.width() / 2+10, display.height() / 2 + 10); - This prints "Starting…", centered on the screen, plus a bit to offset it from the line above.

canvas.pushSprite(0, 0); - Now that we've made all the updates we're going to, we copy the canvas to the LCD at position (0,0). This updates the LCD all at once.

delay(1000); - This causes us to wait one second (1,000 milliseconds).

These lines start the two libraries for the sensors in our sensor array. They specify Wire1, the first alternate I2C bus, the default I2C addresses of those sensors, and I2C pins that the Wire1 bus are connected to on the microcontroller. Then they assign the return code of each operation to the variables qmpPresent and shtPresent, which tells us whether the sensor is present:

```
qmpPresent = qmp.begin(&Wire1, QMP6988_SLAVE_ADDRESS_L, i2cSdaPin, i2cSclPin, 100000U);

shtPresent = sht3x.begin(&Wire1, SHT3X_I2C_ADDR, i2cSdaPin, i2cSclPin, 100000U);
```

Now we're running in the main loop.

This causes the main loop to run once every second.

```
static uint32_t lastFrameMs = 0;

const uint32_t frameIntervalMs = 1000;

if (millis() - lastFrameMs < frameIntervalMs) {

  return;

}

lastFrameMs = millis();
```

This checks to see if the sensor is still present and then calls the sensor library to update the variables with the updated sensor data. When both have completed, the results of the new sensor data is assigned to shtTempF and shtHumidity and qmpTemp and ampPressureInHg and qmpAltitudeFt.

```
if (shtPresent && sht3x.update()) {

  shtTempF = (sht3x.cTemp * 1.8f) + 32.0f;

  shtHumidity = sht3x.humidity;

}

if (qmpPresent && qmp.update()) {

  qmpTempF = (qmp.cTemp * 1.8f) + 32.0f;

  qmpPressureInHg = qmp.pressure * 0.0002953f;

  qmpAltitudeFt = qmp.altitude * 3.28084f;

}
```

drawDashboard(); - this calls the function which creates the canvas and then publishes the canvas to the LCD.

When we get to the end of the list, we loop back to the start of the main loop and run it again. The program doesn't end until we reset or restart the microcontroller:

Reference

Core2 Microcontroller Info: https://docs.m5stack.com/en/core/core2_for_aws

Sensor Info: https://docs.m5stack.com/en/unit/envIII

M5GFX Display Library Stuff: https://docs.m5stack.com/en/arduino/m5gfx/m5gfx_functions

M5Canvas Stuff: https://docs.m5stack.com/en/arduino/m5gfx/m5gfx_canvas